Методические указания по выполнению лабораторной работы №1
«Архитектура и обучение глубоких нейронных сетей»

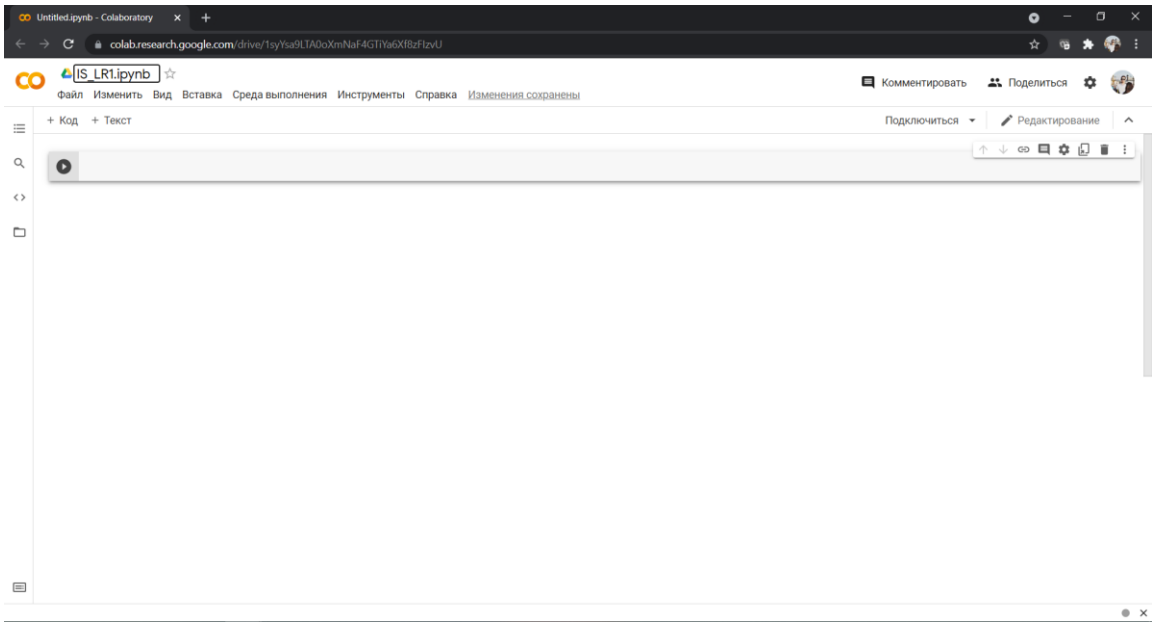
Введение в Google Colab

В курсе лабораторных работ по дисциплине «Интеллектуальные системы», который открывает данная лабораторная работа, рекомендуется пользоваться интерактивной средой программирования Google Colaboratory (https://colab.research.google.com/notebooks/intro.ipynb). Данная среда позволяет писать и выполнять код на Python прямо в браузере. Если вы уже знакомы со средой Jupyter Notebook, то среда Colab будет для вас интуитивно понятной. Если нет – по ссылке выше находится краткое вводное руководство по использованию данной среды, ознакомьтесь с ним прежде всего. Преимуществом работы в Colab, по сравнению с локальной установкой Python и Jupyter Notebook на свой компьютер, является то, что в Colab инженеры Google уже позаботились об установке всевозможных пакетов, в том числе необходимых нам для выполнения курса лабораторных работ, а также о совместимости версий различных пакетов между собой. К тому же, при работе в Colab пользователю предоставляется бесплатный доступ к графическим процессорам, вычисления на которых могут существенно ускорять процесс создания моделей машинного обучения.

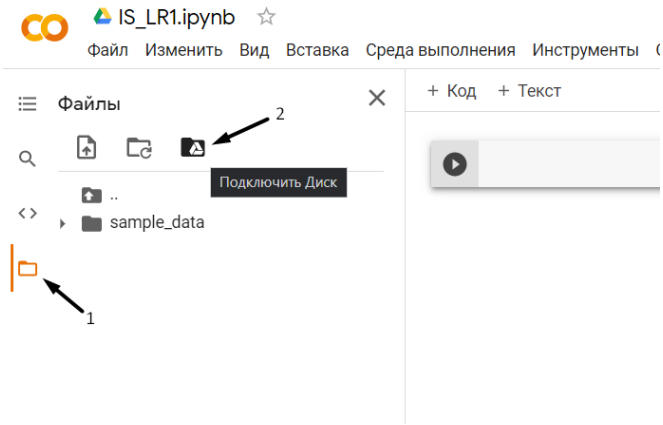
Для работы в Google Colab потребуется аккаунт Google. Создайте аккаунт, если у вас его нет.

Обратите внимание: при работе в Colab длительный простой блокнота приведет к его автоматическому отключению без сохранения переменных.

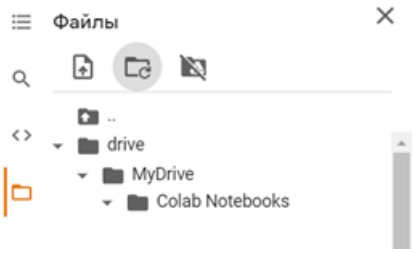
Итак, для создания нового блокнота Colab необходимо воспользоваться меню «Файл» - «Создать блокнот». Открывшийся новый блокнот можно сразу переименовать, нажав на его название в верхней части страницы рядом со значком Google Drive.



Блокноты Colab автоматически сохраняются на вашем Google Drive (далее - диск) в папке Colab Notebooks. Для того, чтобы иметь доступ своему диску из блокнота, требуется разрешить доступ, для чего в левом меню нужно нажать на пиктограмму папки (1), а затем на пиктограмму папки со значком диска (2):



После этого необходимо подтвердить действие «Разрешить этому блокноту доступ к вашим файлам на Google Диске?». В результате в меню слева появится папка «drive», а в ней папка «My Drive» с содержимым вашего Диска:



Зададим в качестве текущей директории ту, в которой будет находиться наш блокнот (вставьте и выполните данный код в новой ячейке):

```
import os
os.chdir('/content/drive/MyDrive/Colab Notebooks')
```

Импорт библиотек и модулей

Основной библиотекой, которой мы будем пользоваться в данном курсе, будет библиотека TensorFlow от компании Google (https://www.tensorflow.org/learn). Данная библиотека – мощный инструмент по разработке моделей машинного обучения. В состав библиотеки TensorFlow входит также библиотека Keras (https://keras.io/about/). Keras является надстройкой над TensorFlow, позволяющей создавать и обучать модели искусственных нейронных сетей за всего несколько строк кода. TensorFlow является для Keras так называемым «вычислительным бэкендом», то есть выполняет работу по созданию и обучению нейронных сетей «под капотом». Помимо перечисленных нам также потребуются вспомогательные библиотеки: NumPy (https://numpy.org/doc/stable/) – для работы с многомерными массивами и выполнения операций над ними, Matplotlib (https://matplotlib.org/stable/contents.html) – для построения графиков, Scikit-learn

(http://scikit-learn.org) – для реализации различных методов обработки данных и алгоритмов машинного обучения и другие.

Программный код для импорта указанных библиотек может выглядеть следующим образом:

```
# импорт модулей
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
import sklearn
```

Загрузка и рассмотрение набора данных

В данной работе мы будем заниматься решением задачи распознавания рукописных цифр. Существует набор данных MNIST, содержащий 70000 изображений рукописных цифр размером 28 на 28 пикселей. Значение каждого пикселя – интенсивность в градациях серого от 0 до 255. Набор данных размечен, то есть каждому изображению поставлена в соответствие метка истинной цифры. Набор MNIST является классическим набором данных для машинного обучения, поэтому в Keras существует метод для его загрузки.

```
# загрузка датасета
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Приведенная выше конструкция автоматически загружает набор данных с сервера и сразу разбивает его на обучающие данные (кортеж (`X_train`, `y_train`)) и тестовые данные (кортеж (`X_test`, `y_test`)) в соотношении 60000:10000 элементов. Однако разбиение не носит характера случайности. Для того, чтобы у каждого студента набор обучающих и тестовых данных оказался немного своим, объединим обучающие и тестовые данные в единый набор (сконкатенируем массивы), а затем разобьем его снова, но уже случайным образом. Для этого нам потребуется функция `train_test_split` из библиотеки sklearn:

```
# создание своего разбиения датасета
from sklearn.model_selection import train_test_split

# объединяем в один набор
X = np.concatenate((X_train, X_test))
y = np.concatenate((y_train, y_test))

# разбиваем по вариантам
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 10000,
                                                    train_size = 60000,
                                                    random_state = 123)
```

Параметр `random_state` задает инициализатор случайного распределения. Это значит, что при заданном значении `random_state` каждый раз будет получаться одно и то же распределение. Использование этого параметра позволяет проводить воспроизводимые опыты.

Вывести размерность массива можно с помощью свойства `shape`:

```
# вывод размерностей
print('Shape of X train:', X_train.shape)
print('Shape of y train:', y_train.shape)
```

Вывести на экран изображение можно с помощью функции pyplot imshow, указав также в качестве параметра цветовую карту в градациях серого. Для вывода метки цифры достаточно просто вызвать print:

```
# вывод изображения
plt.imshow(X_train[123], cmap=plt.get_cmap('gray'))
plt.show()

# вывод метки для этого изображения
print(y_train[123])
```

Предобработка данных

В исходном наборе данных изображения представляются в виде двумерных массивов размером 28 на 28 с целыми значениями интенсивности каждого пикселя от 0 до 255. В таком виде данные не подходят для подачи на вход нейронной сети (по крайней мере такого типа, с которым вы уже знакомы). Для того, чтобы входные изображения можно было подать на вход нейронной сети, их нужно «вытянуть в цепочку», то есть изображение 28 на 28 должно стать вектором длиной $28^2 = 784$ элемента. Также нужно произвести нормировку данных: привести входные значения из диапазона [0;255] к значениям из диапазона [0;1]. Воспользуемся методом reshape, а затем разделим значения на 255, посмотрим на получившуюся размерность:

```
# развернем каждое изображение 28*28 в вектор 784
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels) / 255
X_test = X_test.reshape(X_test.shape[0], num_pixels) / 255

print('Shape of transformed X train:', X_train.shape)
```

Выходные значения (метки цифр) тоже требуют предобработки. Выходные метки являются значениями категориального типа. При решении задачи классификации при помощи нейронных сетей метки классов обычно кодируют по принципу «one-hot encoding». В этом случае каждая метка представляется в виде вектора, длина которого

равна количеству классов (в нашем случае 10), и все элементы которого равны нулю, за исключением одного элемента, равного единице, стоящего на позиции номера этой метки (не забываем, что в Python массивы нумеруются с нуля). Пример:

$$0 \rightarrow [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$5 \rightarrow [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$$

$$3 \rightarrow [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$$

Для перевода используется функция np_utils.to_categorical из модуля utils в Keras. Преобразуем выходные данные и посмотрим на получившуюся размерность, а заодно сохраним в переменную количество классов:

```
# переведем метки в one-hot
from keras.utils import np_utils

y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

print('Shape of transformed y train:', y_train.shape)
num_classes = y_train.shape[1]
```

Реализация модели нейронной сети, и оценка качества классификации

Для реализации модели нейронной сети нам понадобится два класса из Keras. Первый из них – класс Sequential – последовательная модель. Когда мы создаем модель класса Sequential мы последовательно наполняем её слоями нейронов, и сигналы между слоями распространяются также последовательно от входного слоя к выходному. Это базовый и самый распространенный класс моделей, которые позволяет строить Keras. Еще один класс, который нам нужно импортировать, это класс Dense – класс, описывающий полносвязный слой нейронов. Полносвязный означает, что если мы добавим в модель Sequential два слоя Dense, то нейроны в этих двух слоях будут связаны «каждый с каждым».

```
from keras.models import Sequential
from keras.layers import Dense
```

Для примера приведем и разберем описание нейронной сети с двумя скрытыми и одним выходным слоем.

```
# 1. создаем модель - объявляем ее объектом класса Sequential
model = Sequential()

# 2. добавляем первый скрытый слой
model.add(Dense(units=300, input_dim=num_pixels, activation='sigmoid'))
# 3. добавляем второй скрытый слой
model.add(Dense(units=100, activation='sigmoid'))
# 4. добавляем выходной слой
```

```
model.add(Dense(units=num_classes, activation='softmax'))

# 5. компилируем модель
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

Рассмотрим подробно, что тут было сделано.

1. Первым делом, объявили модель объектом класса Sequential.
2. Затем мы добавили в модель первый скрытый слой с помощью метода `add`. Основные параметры слоя – это количество нейронов (`units`) и функция активации (`activation`), однако для первого слоя также обязательно должна быть указана размерность входа – параметр `input_dim`. Этот параметр описывает, какой размерности данные будут поступать на вход первого слоя. В нашем случае на вход сети поступает развернутое в вектор изображение, поэтому входная размерность равна количеству пикселей.
3. Затем добавили в модель второй скрытый слой. Описание второго слоя подобно описанию первого скрытого слоя, за исключением того, что не нужно указывать входную размерность: Keras сам примет для второго слоя входную размерность, равную количеству нейронов в первом слое.
4. Последним добавляется выходной слой. Ему тоже не нужно указывать входную размерность, однако количество нейронов в нем должно совпадать с размерностью выходных данных: в данном случае оно равно количеству классов `num_classes`.
5. После того, как модель наполнена слоями, нужно ее скомпилировать. При компиляции указывается функция ошибки (`loss`), алгоритм оптимизации (`optimizer`) и список метрик, по которым мы будем оценивать качество работы модели (`metrics`).

Для создания однослойной нейронной сети достаточно описать только выходной слой, однако нельзя забыть, что он же является и первым, а значит для него нужно описать входную размерность.

!!! При проведении экспериментов по подбору архитектуры нейронной сети крайне желательно всякий раз задавать новое имя объекта-модели. Простое имя `model` приведено здесь лишь для примера. Используйте численные обозначения `model_1`, `model_2` и т.д. и протоколируйте, какая модель имеет какую архитектуру.

Либо можно использовать более осмысленные названия моделей. Например, для модели с двумя скрытыми слоями, которая была разобрана в примере, можно задать имя `model_1h300_2h100`.

Не забывайте указывать корректное имя модели везде, где вы к ней обращаетесь: при вызове методов `add`, `compile`, и других, которые будут разобраны далее.

Для отображения информации об архитектуре нейронной сети следует воспользоваться методом `summary`. Функция выведет таблицу с информацией о слоях сети: имя слоя и его тип, выходная размерность, количество параметров (весовых коэффициентов).

```
# вывод информации об архитектуре модели
print(model.summary())
```

После того, как мы собрали и скомпилировали модель, ее остается только обучить. Делается это при помощи метода `fit`.

```
# Обучаем модель
H = model.fit(X_train, y_train, validation_split=0.1, epochs=100)
```

В метод `fit` передается массив обучающих входных данных `X_train`, массив обучающих выходных данных `y_train`, количество эпох обучения `epochs`, а также в качестве дополнительного параметра может быть передан параметр `validation_split`. Это значение показывает, какая доля обучающих данных будет отложена и не будет участвовать настройке параметров модели, однако на отложенных данных каждую эпоху также будет вычисляться значение функции ошибки и метрики качества. Отложенные данные называются валидационными. Они используются для контроля процесса обучения. Отслеживание функции ошибки на валидационных данных позволяет контролировать переобучение модели.

По окончании обучения метод `fit` возвращает не только обученную модель, но и специальный объект-отклик (callback), в котором хранятся записи значений функции ошибки и метрики качества на обучающих и валидационных данных за время обучения. Эти значения можно использовать для отображения графика изменения функции ошибки по эпохам:

```
# вывод графика ошибки по эпохам
plt.plot(H.history['loss'])
plt.plot(H.history['val_loss'])
plt.grid()
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.legend(['train_loss', 'val_loss'])
plt.title('Loss by epochs')
plt.show()
```

После того, как модель обучена, остается оценить качество её работы на тестовых данных. Для этого используется функция `evaluate`. Функция вернет список, состоящий из значения функции ошибки и метрики качества на тестовых данных.

```
# Оценка качества работы модели на тестовых данных
scores = model.evaluate(X_test, y_test)
print('Loss on test data:', scores[0])
```

```
print('Accuracy on test data:', scores[1])
```

Метрика качества классификации accuracy (точность) показывает, сколько объектов из тестового множества модель классифицировала правильно по отношению к общему количеству объектов тестового множества.

Сохранение модели на диск и загрузка с диска

Обученную модель можно сохранить на диск для последующего использования. Для этого существует метод `save`. Обязательным аргументом функции является путь, куда будет сохранена модель. В результате сохранения создастся папка с указанным названием, в которой будет сохранена модель.

```
# сохранение модели на диск, к примеру, в папку best_model
# В общем случае может быть указан произвольный путь
model.save('best_model')
```

Для загрузки обученной и сохраненной модели с диска существует метод `load_model`. Он работает обратно методу `save`: обязательным аргументом является путь к папке с моделью.

```
# Загрузка модели с диска
from keras.models import load_model

model = load_model('best_model')
```

Применение обученной модели

Целью создания и обучения нейронной сети в данной работе в конечном счете является возможность ее применения по назначению: для распознавания рукописных цифр.

Для вывода результата обработки нейронной сетью одного изображения воспользуемся функцией `predict`. Выходной слой нейронной сети состоит из 10 нейронов с функцией активации softmax. Особенность данной функции активации в том, что сумма выходных значений всех нейронов равна единице, а значение каждого нейрона можно интерпретировать как вероятность или уверенность нейронной сети в том, что входное значение относится к данному классу. Для определения результата распознавания необходимо воспользоваться функцией `argmax`, то есть определить на каком месте стоит нейрон, выдающий наибольшее значение.

```
# вывод тестового изображения и результата распознавания
n = 123
result = model.predict(X_test[n:n+1])
print('NN output:', result)
```

```
plt.imshow(X_test[n].reshape(28,28), cmap=plt.get_cmap('gray'))
plt.show()
print('Real mark: ', str(np.argmax(y_test[n])))
print('NN answer: ', str(np.argmax(result)))
```

Обратите внимание, что при вызове функции predict на вход сети должен подаваться вектор, имеющий две размерности, поэтому следует воспользоваться срезом (slice) `X_test[n:n+1]`, а не просто `X_test[n]`, который бы был одномерным вектором.

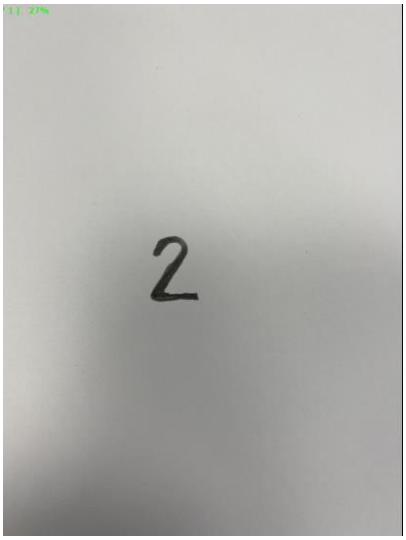
Создание собственного изображения

Для создания собственного экземпляра рукописной цифры требуется взять белый лист бумаги и при помощи темной (черной или синей) ручки или карандаша написать на нем собственную цифру. Цифра должна быть довольно крупной (порядка 2-3 см в высоту), а начертание достаточно жирным. Сфотографируйте собственную цифру с хорошим освещением, чтобы лист на фото был белым, и перенесите фотографию на компьютер.

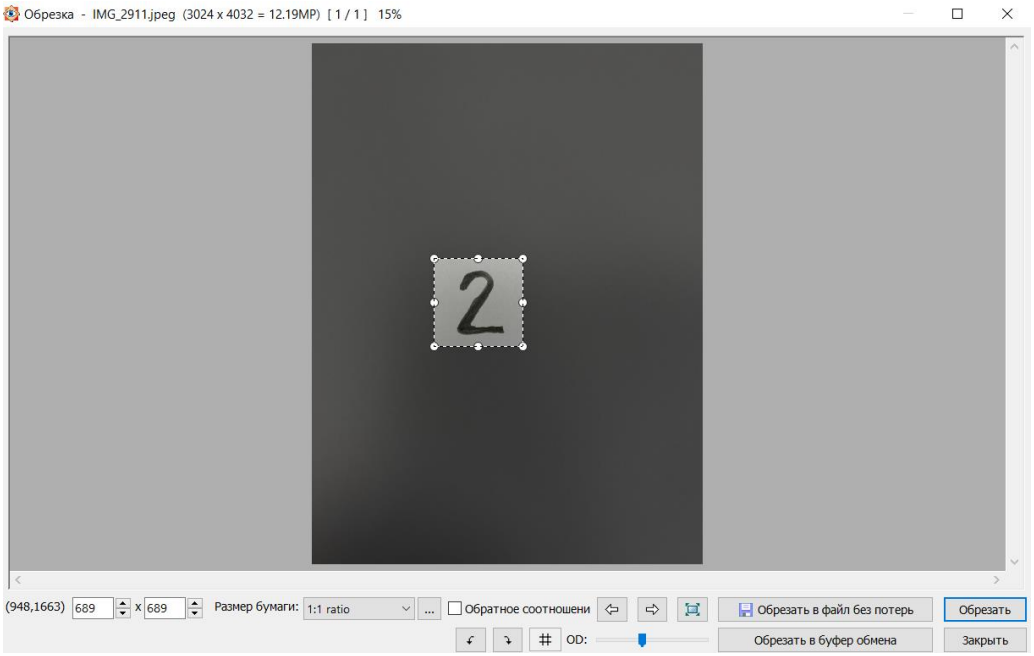
Теперь нужно обработать фотографию так, чтобы изображение цифры стало похожим на изображение в наборе MNIST. Для этого потребуется воспользоваться редактором изображений. Дальнейшие действия будут выполняться в бесплатном редакторе FastStone Image Viewer (скачать: https://www.faststone.org/FSIVDownload.htm).

Запустим редактор и откроем полученную фотографию двойным щелчком по ней (если ваша фотография в формате .heic и редактор не поддерживает данный формат, то для перевода в .jpg можно воспользоваться сервисом https://heic2jpeg.com/ru).

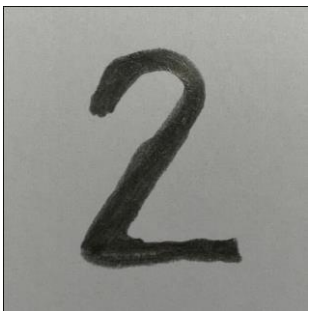
Исходное изображение:



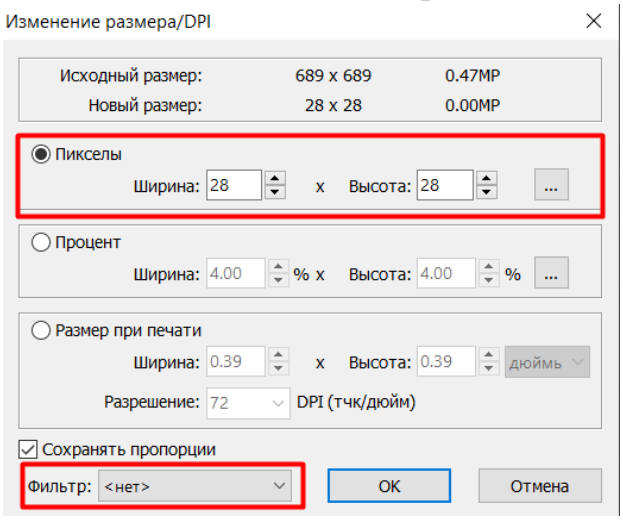
1. Обрежем фото в квадрат (ПКМ – Изменить – Обрезка – Размер бумаги – 1:1 ratio – выделить цифру и сделать ее посередине рамки – обрезать):



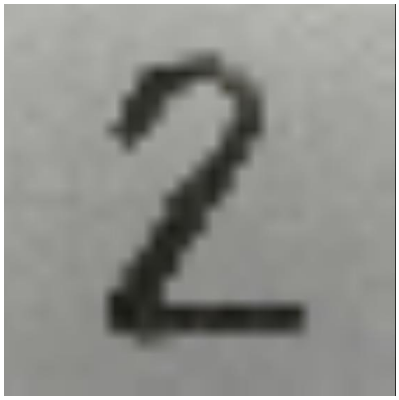
Результат:



2. Приведем изображение к разрешению 28 на 28 пикселей (ПКМ – Изменить – Изменить размер – Пикселы – 28 на 28 – Фильтр – Нет – ОК):



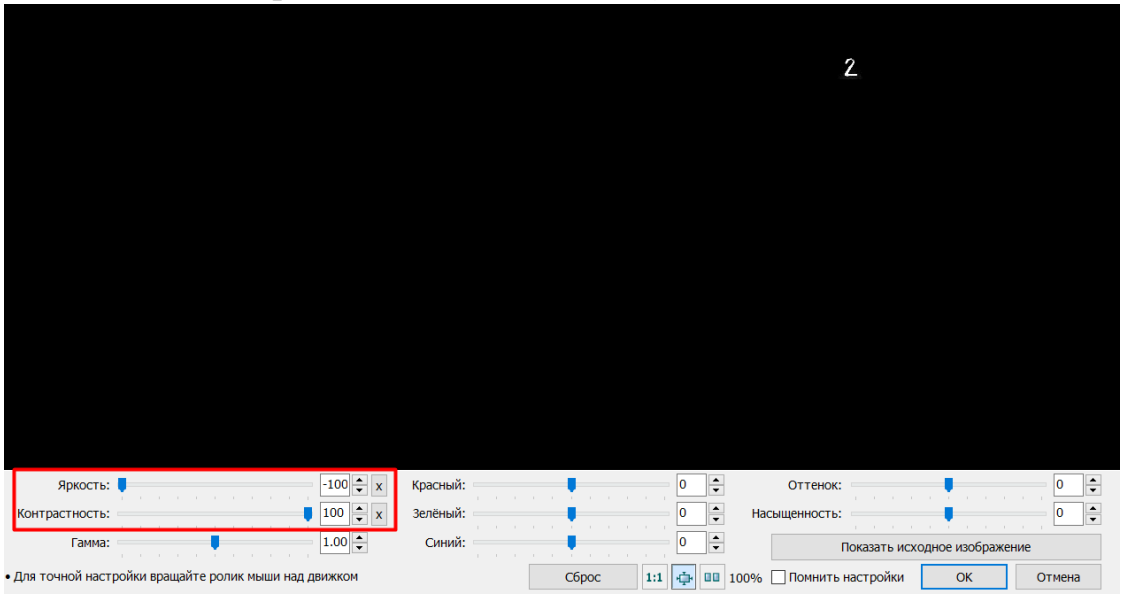
Результат:



3. Переведем изображение в негатив в градациях серого с глубиной цвета 8 бит:
   1. (ПКМ – Цвета – Оттенки серого),
   2. (ПКМ – Цвета – Уменьшить количество цветов – 256 цветов (8 бит))
   3. (ПКМ – Цвета – Негатив)

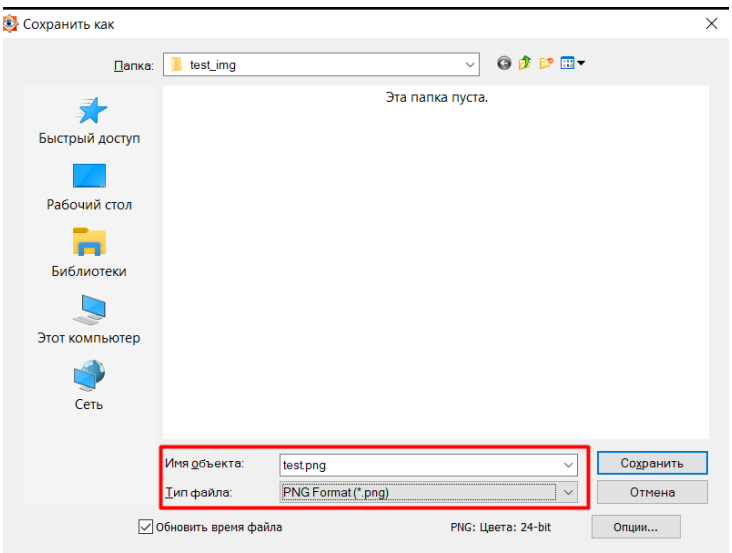
   Результат:

4. Скорректируем яркость и контрастность (ПКМ – Цвета – Коррекция цветов – Яркость: «–100», Контрастность: «+100» – ОК):



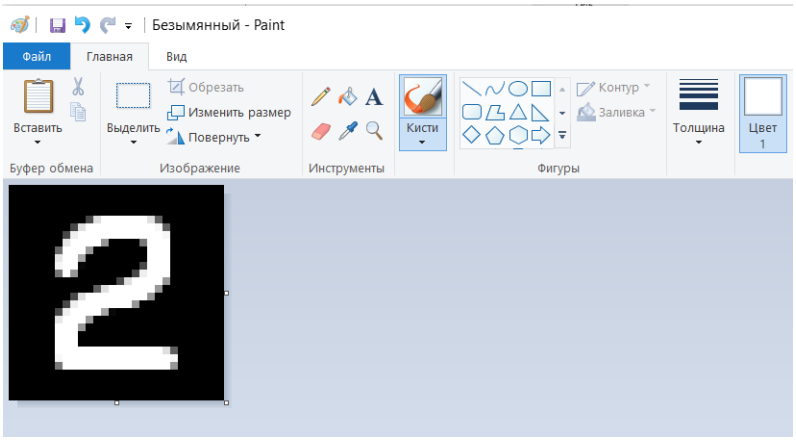
Результат:

5. Сохраним файл в формате .png (Сохранить как – тип файла: PNG Format – Сохранить):



В случае возникновения затруднений с обработкой фотографии в редакторе собственное изображение можно создать в Paint:

1. Залить холст черным цветом.
2. Изменить размер в пикселях на 28 на 28.
3. Инструментом «Кисть» белым цветом толщиной 2 нарисовать цифру.
4. Сохранить изображение в формате .png.



Полученный файл необходимо загрузить на свой Google Диск в папку, к которой мы открыли доступ в начале работы (Мой Диск/Colab Notebook).

Для загрузки в программу изображения как массива пикселей воспользуемся библиотекой PIL (Python Image Library):

```
# загрузка собственного изображения
from PIL import Image
file_data = Image.open('test.png')
file_data = file_data.convert('L') # перевод в градации серого
test_img = np.array(file_data)
```

Остается только отобразить загруженное изображение, предобработать его и подать на вход нейронной сети, а затем вывести результат распознавания:

```
# вывод собственного изображения
plt.imshow(test_img, cmap=plt.get_cmap('gray'))
plt.show()

# предобработка
test_img = test_img / 255
test_img = test_img.reshape(1, num_pixels)

# распознавание
result = model.predict(test_img)
print('I think it\'s ', np.argmax(result))
```